- **生成二进制文件**
  - 生成目标文件
    - 命令：
      - $gcc –c hello.c –o hello.o
  - 生成可执行文件
    - 命令：
      - $gcc hello.c –o hello
      - 运行程序
      - $./hello
        hello gcc!
- **生成汇编文件**
  - 命令
    - $gcc –S hello.c –o hello.s
- **生成预处理文件**
  - 命令
    - $gcc –E hello.c –o hello.i

gcc文件扩展名规范		
扩展名	类型	可进行的操作方式
.c	c语言源程序	预处理、编译、汇编、链接
.C，.cc，.cp，.cpp，.c++，.cxx	c++语言源程序	预处理、编译、汇编、链接
.i	预处理后的c语言源程序	编译、汇编、链接
.ii	预处理后的c++语言源程序	编译、汇编、链接
.s	预处理后的汇编程序	汇编、链接
.S	未预处理的汇编程序	预处理、汇编、链接
.h	头文件	不进行任何操作
.o	目标文件	链接

gcc常用选项	
选项	含义
-c	仅对源文件进行编译，不链接生成可执行文件。在对源文件进行查错时，或只需产生目标文件时可以使用该选项。
-g[gdb]	在可执行文件中加入调试信息，方便进行程序的调试。如果使用中括号中的选项，表示加入gdb扩展的调试信息，方便使用gdb来进行调试
-O[0、1、2、3]	对生成的代码使用优化，中括号中的部分为优化级别，缺省的情况为2级优化，0为不进行优化。注意，采用更高级的优化并不一定得到效率更高的代码。
-D*name[=definition]*	将名为*name*的宏定义为*definition*，如果中括号中的部分缺省，则宏被定义为1

选项	含义
-I*dir*	在编译源程序时增加一个搜索头文件的额外目录——*dir*，即include增加一个搜索的额外目录。
-L*dir*	在编译源文件时增加一个搜索库文件的额外目录——*dir*
-l*library*	在编译链接文件时增加一个额外的库，库名为*library*.a
-w	禁止所有警告
-W*warning*	允许产生*warning*类型的警告，*warning*可以是：main、unused等很多取值，最常用是-Wall，表示产生所有警告。如果*warning*取值为error，其含义是将所有警告作为错误（error），即出现警告就停止编译。

- 目录结构(1)
  - 编译命令

$ gcc *my_app.c greeting.c* –o my_app

- 目录结构(2)
  - 编译方式(1)

$ gcc my_app.c *functions/greeting.c* –o my_app *-I function*

Makefile文件	
1	my_app:greeting.o my_app.o
2	gcc my_app.o greeting.o -o my_app
3	greeting.o:functions\greeting.c functions\greeting.h
4	gcc -c functions\greeting.c
5	my_app.o:my_app.c functions\greeting.h
6	gcc –c my_app.c –I functions

分块编译

- greeting.c
  - $gcc *-g -Wall* -c functions/greeting.c
    - -g：将调试信息加入到编译的目标文件中；
    - -Wall：将编译过程中的所有级别的警告都打印出来；
  - 无错误
- my_app.c
  - $gcc *-g -Wall* -c my_app.c -Ifunctions
  - 参数含义同上
  - 错误信息：

更实用的Makefile文件	
1	OBJS = greeting.o my_app.o
2	CC = gcc
3	CFLAGS = -Wall -O –g
4	my_app:${OBJS}
5	${CC} ${OBJS} -o my_app
6	greeting.o:functions\greeting.c functions\greeting.h
7	${CC} ${CFLAGS} -c functions\greeting.c
8	my_app.o:my_app.c functions\greeting.h
9	${CC} ${CFLAGS} -c my_app.c -Ifunctions

gdb常用的调试命令	
命令	含义
file	指定需要进行调试的程序
step	单步（行）执行，如果遇到函数会进入函数内部
next	单步（行）执行，如果遇到函数不会进入函数内部
run	启动被执行的程序
quit	退出gdb调试环境
print	查看变量或者表达式的值
break	设置断点，程序执行到断点就会暂停起来
shell	执行其后的shell命令
list	查看指定文件或者函数的源代码，并标出行号

杀死进程

- kill
  - 格式 :kill[选项]进程号
  - **-l 信号，若果不加信号的编号参数，则使用“-l”参数会列出全部的信号名称**
  - **-a 当处理当前进程时，不限制命令名和进程号的对应关系**
  - **-p 指定kill 命令只打印相关进程的进程号，而不发送任何信号**
  - **-s 指定发送信号**
  - **-u 指定用户**

范例：

kill –s SIGKILL 4096

杀死4096号进程。

- ps
  - 格式 :ps[选项]
  - 范例：
  - ps aux
  - 查看系统中所有进程。

软件安装

- rpm
  - 格式 :rpm [选项][安装文件]
  - 范例：
  - 1. rpm –ivh tftp.rpm
  - 安装tftp
  - 2.rpm –qa
  - 列出所有已安装的rpm包
  - 3. rmp –e name
  - 卸载名为name的rpm包

常用命令

- 显示文字命令echo

  echo [-n] <字符串>

- 显示日历命令cal

  cal [选项] [[月] 年]

- 日期时间命令date

  显示日期和时间的命令格式为：

  date [选项] [+FormatString]

  - 设置日期和时间的命令格式为：

    date <SetString>

- 清除屏幕命令clear

访问权限

- **每一文件或目录的访问权限都有3种，每组用三位表示，分别为文件所有者的读、写和执行权限；与所有者同组的用户的读、写和执行权限；系统中其他用户的读、写和执行权限。当用ls –l命令显示文件的详细信息时，最左边的一列为文件的访问权限。**

■ chmod

- 格式 :chmod [who] [+|-|=][mode]文件名
- 参数
- Who
- u 表示文件的所有者
- g 表示与文件的所有者同组的用户
- o 表示其他用户
- a 表示所有用户，它是系统默认值
- Mode：
- + 添加权限， - 取消权限， =赋予权限
- 如： chmod g + w hello.c

- **Mode所表示的权限可使用8进制数表示**
- **r=4, w=2, x=1，分别以数字之和表示权限**
- 范例:

  chmod 761 hello.c

  chmod 777 hello

打包与压缩

■ tar

- 格式 :tar [选项] 目录或文件
- 范例
- 1. tar cvf tmp.tar /home/tmp
- 将/home/tmp目录下所有文件与目录打包成一个tmp.tar文件
- 2. tar xvf tmp.tar
- 将打包文件tmp.tar在当前目录下解开
- 3. tar cvzf tmp.tar.gz /home/tmp
- 将/home/tmp目录下所有文件与目录打包并压缩成一个tmp.tar.gz文件
- 4. tar xvzf cvf tmp.tar.gz
- 将打包压缩文件tmp.tar.gz在当前目录下解开

查看目录

■ **ls**

- **格式 :ls [选项] [目录或文件]**
- **范例**
- **1. ls /home**
- **显示/home目录下文件与目录(不包含隐藏文件)**
- **2. ls –a /home**
- **显示/home目录下所有文件与目录(包含隐藏文件)**

查看当前路径

■pwd

- 格式 :pwd
- 范例
- pwd
- 显示当前工作目录的绝对路径

删除

■ **rm**

- **格式 :rm [选项] 源文件或目录**
- **范例**
- **1. rm /home/test**
- **将/home目录下test文件删除**
- **2. rm –r /home/lky**
- **将/home目录下的lky目录删除**

创建目录

■ **mkdir**

- **格式 :mkdir [选项] 目录名**
- **范例**
- **1. mkdir /home/test**
- **在/home目录下创建tes目录**
- **2. mkdir –p /home/lky/tmp/**
- **创建/home/lky/tmp目录，如果lky不存在，先创建lky**

移动或更名

■ **mv**

- **格式 :mv [选项] 源文件或目录 目标文件或目录**
- **范例**
- **1. mv /home/test /home/test1**
- **将/home目录下test文件更名为/tmp目录下**
- **2. mv /home/lky /tmp/**
- **将/home目录下的lky目录移动（剪切）到/tmp目录下**

■ **cp**

- **格式 :cp [选项] 源文件或目录 目标文件或目录**
- **范例**
- **1. cp /home/test /tmp**
- **将/home目录下test文件copy到/tmp目录下**
- **2. cp –r /home/lky /tmp/**
- **将/home目录下的lky目录copy到/tmp目录下**

关机

- shutdown
- 格式 shutdown [-t seconds] [-rkhncfF] time [message]
- 范例
- shutdown now
- 立即关机

切换用户

- su
- 格式 su [选项] [用户名]
- 范例
- su -root
- 切换到root用户，并将root的环境变量同时代入

增加用户

- useradd
- 格式 useradd [选项] 用户名
- 范例
- useradd sjg
- 添加名字为sjg的用户

- cmd [-参数] [操作对象]
  - cmd是命令名
  - 单字符参数前使用一个减号（-），单词参数前使用两个减号（--）。
  - 多个单字符参数前可以只使用一个减号。
  - 最简单的Shell命令只有命令名，复杂的Shell命令可以有多个参数。
  - 操作对象可以是文件也可以是目录，有些命令必须使用多个操作对象， 如cp命令必须指定源操作对象和目标操作对象。
  - 命令名、参数和操作对象都作为Shell命令执行时的输入，它们之间用空格分隔开。

■ **Linux是一个多用户操作系统，它可以同时接受多个用户登录。 Linux还允许一个用户进行多次登陆，因为Linux提供了虚拟控制台的访问方式。**

■ 一般从图形界面--〉文本界面

– **Ctrl+Alt+Fn n=1-6**

■ 文本界面--〉图形界面

– **Alt+F7**

■ 命令行技巧

Tab键可补全命令
上下键调用命令历史记录

■ 命令的终止

大部分命令，ctrl+c可终止执行
部分命令，例如man，用“q”退出

■ Shell的种类

– ash：是贝尔实验室开发的shell，bsh是对ash的符号链接。
– bash：是GNU的Bourne Again shell，是GNU默认的shell。sh以及bash2都是对它的符号链接。
– tcsh：是Berkeley UNIX C shell。csh是对它的符号链接 。

■ Shell的主要功能

命令解释器、命令通配符、命令补全、别名机制、命令历史

■ **五个主要子系统**

(1)进程调度
(2)进程间通信
(3)内存管理
(4)虚拟文件系统
(5)网络接口

■ **其他部分**

(6) 各子系统需要对应的设备驱动程序
(7) 依赖体系结构的代码

本质不同：Windows文件系统只负责文件存 储，Linux文件系统管理所有软硬件资源

进程调度

■进程调度负责控制进程访问CPU

✓保证进程公平地使用CPU

✓ 保证内核能够准时执行一些必要的硬件操作

■所有其他子系统都依赖于进程调度

■Linux采用基于优先级的进程调度方法

✓SCI（系统调用接口） 层提供了某些机制执行 从用户空间到内核的函数调用

✓通过优先级确保重要进程优先执行

进程间通信

■ 进程间通信就是在不同进程之间传播或交换信息

■ 每个进程各自有不同的用户地址空间，进程间并不能直接通信，所以进程之间要交换数据必须通 过内核

■ 内核中开辟一块缓冲区，进程1把数据从用户空间拷到内核缓冲区，进程2再从内核缓冲区把数据读走，内核提供的这种机制称为进程间通信（IPC，InterProcess Communication）

内存管理

■ 内存管理可以使多个进程安全地共享内存

IPC中的共享内存方式依赖于内存管理

■ 管理虚拟内存

Linux 包括了管理可用内存的方式，以及物理 和虚拟映射所使用的硬件机制。

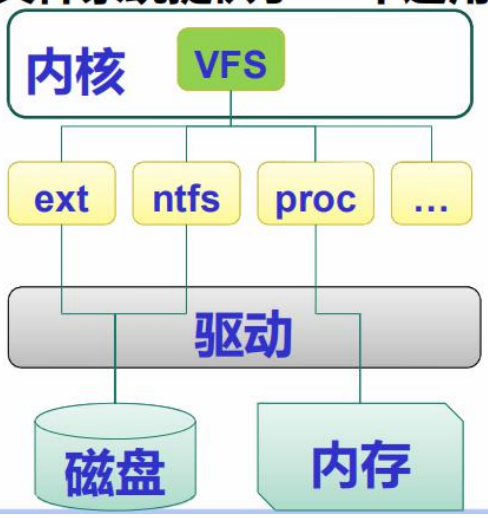
虚拟文件系统

- 虚拟文件系统（VFS）为文件系统提供了一 个通用的接抽象
- VFS

  逻辑文件系统

  EXT4、NFS等

  设备驱动程序



网络接口

- 网络接口在设计上遵循模拟协议本身的分层体系结构
  - ✓ 网络协议
  - ✓ 网络设备驱动程序
- Linux网络支持TCP/IP模型
  - ✓ IP协议
  - ✓ TCP协议
  - ✓ UDP协议

- Linux系统中有三种基本的文件类型
  - 普通文件：又分为文本文件和二进制文件；
  - 目录文件：目录文件存储了一组相关文件的位置、大小等与文件有关的信息；
  - 设备文件：Linux系统把每一个I/O设备都看成一个文件，与普通文件一样处理，这样可以使文件与设备的操作尽可能统一 。

- Linux系统以目录的方式来组织和管理系统中的所有文件

✓ Linux系统通过目录将系统中所有的文件分级、分层组织在一起，形成了Linux文件系统的树型层次结构。以根目录“/”为起点，所有其他的目录都由根目录派生而来。

✓ 特殊目录:“.”代表该目录自己，“..”代表该目录的父目录，对于根目录，“.”和“..”都代表其自己。

什么是交叉编译

- 在一种平台上编译出能在另一种平台（体系结构不同）上运行的程序；
- 在**PC平台(X86)**上编译出能运行在**ARM**平台上的程序，即编译得到的程序在**X86**平台上不能运行，必须放到**ARM**平台上才能运行；
- 用来编译这种程序的编译器就叫交叉编译器；
- 为了不与本地编译器混淆，交叉编译器的名字一般都有前缀，例如：**arm-linux-gcc**。

- 基于ARM体系结构的gcc交叉开发环境中，arm-linux-gcc是交叉编译器，arm-linux-ld是交叉链接器

1、程序编辑

vi debug.c

```
#include <stdio.h>
int func(int n)
{
        int sum = 0,i;
        for(i=0;i<n;i++)
        {
                sum += i;
        }
        return sum;
}

main()
{
        int i;
        long result=0;
        for(i=1;i<=100;i++)
        {
                result+=i;
        }
        printf("result[1-100]=%d \n",result);
        printf("result[1-250]=%d \n",func(250));
}
```

2、程序编译

gcc debug.c -o debug -g

3、程序运行

./debug

4、程序调试

gdb debug

三种内部时钟信号

FCLK 内核时钟	HCLK 总线时钟	PCLK I/O接口时钟
• 供微处理器（内核）使用的时钟信号	• 供外围总线APB使用的时钟信号	• 供高性能总线AHB使用的时钟信号

举例：时钟信号频率计算

△已知时钟源的频率为12MHz，MPLLCON中MDIV=92，PDIV=1，SDIV=1，且CLKDIVN中PCLK:HCLK:FCLK设置为1:2:8。试计算FCLK、HCLK和PCLK的频率。

分频参数：m = (MDIV + 8), p = (PDIV + 2), s = SDIV

=92+8　　=1+2　　=1

FCLK＝2*(92+8)*(12000000)/(3+2^1)=400000000=400MHz

HCLK=400/4=100MHz

PCLK=400/8=50MHz

分频比例的选择

时钟分频控制（CLKDIVN）寄存器

寄存器	地址	R/W	描述	复位值
CLKDIVN	0x4C000014	R/W	时钟分频控制寄存器	0x00000004

CLKDIVN	位	描述	初始状态
DIVN_UPLL	[3]	UCLK 选择寄存器（UCLK 必须为 48MHz 给 USB） 0：UCLK = UPLL 时钟　　1：UCLK = UPLL 时钟 / 2 当 UPLL 时钟被设置为 48MHz 时，设置为 0 当 UPLL 时钟被设置为 96MHz 时，设置为 1	0
HDIVN	[2:1]	00：HCLK = FCLK/1 01：HCLK = FCLK/2 10：HCLK = FCLK/4　当 CAMDIVN[9] = 0 时 HCLK = FCLK/8　当 CAMDIVN[9] = 1 时 11：HCLK = FCLK/3　当 CAMDIVN[8] = 0 时 HCLK = FCLK/6　当 CAMDIVN[8] = 1 时	00
PDIVN	[0]	0：PCLK 是和 HCLK/1 相同的时钟 1：PCLK 是和 HCLK/2 相同的时钟	0

△时钟频率的计算公式

FCLK= $F_{OUT} = 2 * m * Fin / (p*2^S)$

- Fin：输入时钟源的频率。
- m、p、s：分频控制参数，分别由锁相环配置寄存器MPLLCON中MDIV、PDIV和SDIV的值确定。

时钟控制模块(续)

➢举例：相关寄存器的设定程序

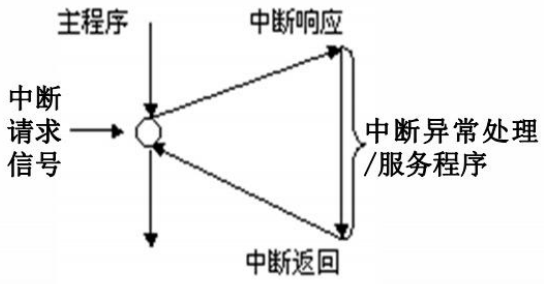
```
;设置锁相环控制寄存器MPLLCON
ldr   r0,  = MPLLCON
ldr   r1,  = ((92 << 12) + (1 << 4) + 1)
str   r1,  [r0]

;设置时钟分频控制寄存器CLKDIVN
ldr   r0,=CLKDIVN
ldr   r1,=0x5          ;0x5=0101b
str   r1,[r0]
```

PLLCON	Bit	
MDIV	[19:12]	Main divider control
PDIV	[9:4]	Pre-divider control
SDIV	[1:0]	Post divider control

CLKDIVN	Bit	Description
DIVN_UPLL	[3]	UCLK select register(UCLK must be 48MHz for USB) 0: UCLK = UPLL clock 1: UCLK = UPLL clock / 2 Set to 0, when UPLL clock is set as 48MHz Set to 1. when UPLL clock is set as 96MHz.
HDIVN	[2:1]	00 : HCLK = FCLK/1. 01 : HCLK = FCLK/2. 10 : HCLK = FCLK/4 when CAMDIVN[9] = 0. HCLK= FCLK/8 when CAMDIVN[9] = 1. 11 : HCLK = FCLK/3 when CAMDIVN[8] = 0. HCLK = FCLK/6 when CAMDIVN[8] = 1.
PDIVN	[0]	0: PCLK has the clock same as the HCLK/1. 1: PCLK has the clock same as the HCLK/2.

中断的概念

➢微处理器在执行正常程序的过程中，因某事件发生，收到来自外围部件的请求信号。若能够响应该信号，则暂停当前程序的正常执行，转去执行针对请求事件的处理操作，待结束后再返回被暂时中断的程序继续执行。



greeting.sh		解释
1	#!/bin/bash	以 #! 开始，其后为使用的shell
2	#a Simple shell Script Example	以 # 开始，其后为程序注释
3	#a Function	同上
4	function say_hello()	以 functio 开始，定义函数
5	{	函数开始
6	echo "Enter Your Name,Please. :"	echo命令输出字符串
7	read name	读入用户的输入到变量name
8	echo "Hello $name"	输出
9	}	函数结束
10	echo "Programme Starts Here....."	程序开始的第一条命令，输出提示信息
11	say_hello	调用函数
12	echo "Programme Ends."	输出提示，提示程序结束

- Shell编程中，使用变量无需事先声明
- 变量的赋值与引用：
  - ✓ 赋值：变量名=变量值（注意：不能留空格）
  - ✓ 引用：$var引用var变量
- Shell变量有几种类型
  - ✓ 用户自定义变量
  - ✓ 环境变量
  - ✓ 位置参数变量
  - ✓ 专用参数变量
- 由用户自己定义、修改和使用，Shell的默认赋值是字符串赋值

```
var=1
var=$var+1
echo  $var          #打印的结果是什么呢?
```

- 为了达到想要的效果有以下几种表达方式
  - ✓ let "var+=1"
  - ✓ var=$[$var+1]
  - ✓ var=`expr $var + 1`#注意加号两边的空格

变量的引用

- **格式:**
  $变量名，或者${变量名}
  变量名为一个字符用方式一，变量名多于一个字符建议用第2种方式
- **例子:**
  a=1
  abc="hello"
  echo $a
  echo ${abc}

■ 单引号

由单引号括起来的字符都作为普通字符出现， 即使是$、`和

echo 'The time is `date` , the file is $HOME/abc '

The time is ` date ` , the file is $HOME/abc

■ 倒引号

倒引号括起来的字符串被shell解释为命令行， 在执行时，Shell会先执行该命令行，并以它的 标准输出结果取代整个倒引号部分。在前面示例中已经见过。

- echo “Dir is `pwd` and logname is $LOGNAME”
- names="Zhangsan Lisi Wangwu"

■ 双引号

- **双引号内的字符，除$、`和\仍保留其特殊功能外，其余字符均作为普通字符对待**
- **$表示变量替换**
- **倒引号表示命令替换**
- **\为转义字符**

- echo “Dir is `pwd` and logname is $LOGNAME”
- names="Zhangsan Lisi Wangwu"

- $[]：可以接受不同基数的数字的表达式

echo $[10+1]（输出：11）

echo “$[2+3],$HOME”（输出：5,/root）

echo $[2<<3],$[8>>1]（输出：16,4）

echo $[2>3],$[3>2]（输出：0,1 表达式为false时输出0，为true时输出1）

- 字符表达式：直接书写，采用单引号，双引号引起来。

echo " $HOME，That is your root directory."（输出：/root，That is your root directory.）

echo ' $HOME，That is your root directory. '（输出：$HOME，That is your root directory.）

单引号和双引号的区别在于：单引号是原样显示，双引号则显示出变量的值。

- **if分支**
  - 格式：

```
if 条件1
then
命令
[elif 条件2
   then
命令]
[else
命令]
fi
```

  - 说明：
    - 中括号中的部分可省略；
    - 当条件为真（0）时执行then后面的语句，否则执行else后面的语句；
    - 以fi作为if结构的结束。

- **#!/bin/bash**

  #if.sh

  if ["10" -lt "12"] **#注意：if和[之间，[和"10"之间，"12"和]都有空格，如果不加空格，会出现语法错误**

  then

  echo "Yes,10 is less than 12"

  fi

case分支

- 格式：

```
case 条件 in
模式1）
        命令1
        ；；
[模式2）
        命令2
        ；；
..............
模式n）
        命令n
        ；；]
esac
```

- 说明：
  - “条件”可以是变量、表达式、shell命令等；
  - “模式”为条件的值，并且一个“模式”可以匹配多种值，不同值之间用竖线（|）联结；
  - 一个模式要用双分号（；；）作为结束；
  - 以逆序的case命令（esac）表示case分支语句的结束

```
#!/bin/bash
#case.sh
echo -n "Enter a start or stop:"
read ANS
case $ANS in
start)
echo "You select start"
;;
stop)
echo "You select stop"
;;
*)
echo "`basename $0`: You select is not between start and stop"
>&2
#注意：>和&2之间没有空格,>&2 表示将显示输出到标准输出（一般是屏幕）上
exit;
;;
esac
```

for循环

- 格式

```
for 变量 [in 列表]
do
        命令（通常用到循环变量）
done
```

- 说明：
  - “列表”为存储了一系列值的列表，随着循环的进行，变量从列表中的第一个值依次取到最后一个值；
  - do和done之间的命令通常为根据变量进行处理的一系列命令，这些命令每次循环都执行一次；
  - 如果中括号中的部分省略掉，Bash则认为是“in $@”，即执行该程序时通过命令行传给程序的所有参数的列表。

```
#!/bin/sh

for foo in bar fud 43
do
  echo $foo
done
exit 0
```

That results in the following output:

```
bar
fud
43
```

```
#!/bin/sh

echo "Enter password"
read trythis

while [ "$trythis" != "secret" ]; do
  echo "Sorry, try again"
  read trythis
done
exit 0
```

An example of the output from this script is as follows:

```
Enter password
password
Sorry, try again
secret
$
```